Lecture 23: Web Services

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

Some slides adapted (and slightly modified) from:
• Clark Barrett
• Jinyang Li
• Randy Bryant
• Dave O’Hallaron
Web History

• 1989:
  – Tim Berners-Lee (CERN) writes proposal on a distributed hypertext system.
    • Connects “a web of notes with links.”
• 1990-92:
  – Tim BL writes a graphical browser for Next machines.
  – NCSA server released
• 1993
  – Marc Andreessen releases first version of NCSA Mosaic browser
  – Mosaic version released for (Windows, Mac, Unix).
• 1994
  – Andreessen founded “Mosaic Communications Corp” (predecessor to Netscape).
Web Communication

- Clients and servers communicate using HyperText Transfer Protocol (HTTP)

- Current version is HTTP/1.1
  - RFC 2616, June, 1999.
Web Services vs FTP

- Web content can be written in a language known as HTML (Hypertext Markup Language).

- An HTML program (page) contains instructions (tags) that tell the browser how to display various text and graphical objects in the page.
Web Content

- Web servers return content to clients
  - content: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

- Example MIME types
  - text/html
    - HTML document
  - text/plain
    - Unformatted text
  - application/postscript
    - Postscript document
  - image/gif
    - Binary image encoded in GIF format
  - image/jpeg
    - Binary image encoded in JPEG format
Static and Dynamic Content

- Content returned in HTTP responses can be static or dynamic.
  - **Static**: content stored in files
    - Examples: HTML files, images, audio clips.
    - Request identifies content file
  - **Dynamic**: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client.
    - Request identifies file containing executable code

- **Bottom line**: All Web content is associated with a file that is managed by the server.
Every piece of content returned by a Web server is associated with some file that it manages.

Each of these files has a unique name known as a URL (Universal Resource Locator).

URLs for static content:
- http://cs.nyu.edu/courses/spring13/CSCI-UA.0201-003/index.html

URLs for dynamic content:
- http://cims.nyu.edu/~th1133/cgi-bin/checkgrade.cgi
http://www.google.com:80/index.html

identifies:
• An html file called index.html
• On internet host www.google.com
• Managed by web server listening to port 80
http://bluefish.ics.cs.cmu.edu:8000/cgi-bin/adder?15000&213

Identifies an executable

Called adder

That has two arguments: 15000 and 213
Anatomy of an HTTP Transaction

unix> telnet www.aol.com 80
Trying 205.188.146.23...
Connected to aol.com.
Escape character is '^[].'

GET / HTTP/1.1
Host: www.aol.com

HTTP/1.0 200 OK
MIME-Version: 1.0
Date: Mon, 8 Jan 2010 4:59:42 GMT
Server: Apache-Coyote/1.1
Content-Type: text/html
Content-Length: 42092

<html>
  ...
</html>

Connection closed by foreign host.
unix>

Client: open connection to server
Telnet prints 3 lines to the terminal

<method> <URI> <version>

Client: request line
Client: required HTTP/1.1 header
Client: empty line terminates headers
Server: response line
Server: followed by five response headers

Server: expect HTML in the response body
Server: expect 42,092 bytes in the response body
Server: empty line terminates response headers
Server: first HTML line in response body
Server: 766 lines of HTML not shown
Server: last HTML line in response body
Server: closes connection
Client: closes connection and terminates
Request Line

• `<Method> <URI> <Version>`
• HTTP supports a number of different methods, including GET, POST, OPTIONS, HEAD, PUT, DELETE, and TRACE.
• **GET method:**
  – accounts for over 99% of HTTP requests
  – instructs the server to generate and return the content identified by the URI (Uniform Resource Identifier).
• The URI is the suffix of the corresponding URL that includes the file name and optional arguments.
Request Headers

- After the request line
- Provide additional information to the server, such as the brand name of the browser or the MIME types that the browser understands.
- Has the form

  `<header name> : <header data>`
HTTP Responses

• Response line followed by one or more response headers

• HTTP Response line:
  \(<version> \ <status \ code> \ <status \ msg>\)
    – \(<version>\) is HTTP version of the response.
    – \(<status \ code>\) is numeric status.
    – \(<status \ msg>\) is corresponding English text.
      • 200  OK          Request was handled without error
      • 301  Moved       Provide alternate URL
      • 403  Forbidden   Server lacks permission to access file
      • 404  Not found   Server couldn’t find the file.

• Response headers:
  \(<header \ name>\): \(<header \ data>\)
Example response headers:
  – Content-Type:  MIME type of content in response body.
  – Content-Length: Length of content in response body.
Data Transfer Mechanisms

• **Standard**
  – Specify total length with content-length

• **Chunked**
  – Break into blocks
  – Prefix each block with number of bytes (Hex coded)
Serving dynamic content

- A web server can invoke a **CGI program** to generate content on the fly

```
Welcome to add.com: THE Internet addition portal.

The answer is: 15213 + 18243 -> 33456.

Thanks for visiting!
```
Serving Dynamic Content With GET

• URL:
  
  - `cgi-bin/adder?n1=15213&n2=18243`

• Result displayed on browser:

  Welcome to add.com: THE Internet addition portal. The answer is: 15213 + 18243 -> 33456
  Thanks for visiting!
Serving Dynamic Content With GET

• How does the client pass arguments to the server?
  – Part of a URL!
  – `http://add.com/cgi-bin/adder?n1=15213&n2=18243`

• URI often generated by an HTML form

```html
<FORM METHOD=GET ACTION="cgi-bin/adder">
  <p>X <INPUT NAME="n1"/>
  <p>Y <INPUT NAME="n2"/>
  <p><INPUT TYPE=submit>
</FORM>
```

URI: Uniform Resource Identifier
Building a simple web server

- Read request from client
- Parse header, split into `<method> <uri> <version>`
- If URI contains "cgi-bin" then serve dynamic content
  - Fork process to execute program
- Otherwise serve static content
  - Copy file to output
Tiny Serving Static Content

/* Send response headers to client */
get_filetype(filename, filetype);
sprintf(buf, "HTTP/1.0 200 OK\r\n");
sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);
sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);
sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);
Rio_writen(fd, buf, strlen(buf));

/* Send response body to client */
srcfd = Open(filename, O_RDONLY, 0);
srcp = Mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0);
Close(srcfd);
Rio_writen(fd, srcp, filesize);
Munmap(srcp, filesize);

- Serve file specified by filename
- Use file metadata to compose header
- “Read” file via mmap
- Write to output

From tiny.c
Serving Dynamic Content

- **Client sends request to server.**
- **If request URI contains the string “/cgi-bin”, then the server assumes that the request is for dynamic content.**

```
GET /cgi-bin/env.pl HTTP/1.1
```
Serving Dynamic Content (cont)

- The server creates a child process and runs the program identified by the URI in that process.
Serving Dynamic Content (cont)

• The child runs and generates the dynamic content.
• The server captures the content of the child and forwards it without modification to the client.
Issues in Serving Dynamic Content

• How does the server pass arguments to the child?
• How does the server capture the content produced by the child?
Serving Dynamic Content With GET

• How does the server pass program arguments to child?
  – In environment variable QUERY_STRING
  – E.g. for adder: \texttt{QUERY\_STRING = “n1=15213\&n2=18243”}

```c
if ((buf = getenv("QUERY\_STRING")) != NULL) {
    if (sscanf(buf, "%d\&%d\n", &n1, &n2) == 2)
        sprintf(msg, "%d + %d \-> %d\n", n1, n2, n1+n2);
    else
        sprintf(msg, "Can't parse buffer '%s'\n", buf);
}
```

From \texttt{adder.c}
Serving Dynamic Content With GET

• How does the server capture the content produced by the child?
  – Child generates its output on stdout. Server redirects stdout to its connected socket.
  – The child must generate the corresponding HTTP response headers.

/* Make the response body */
printf(content, "Welcome to add.com: ");
printf(content, "%sTHE Internet addition portal.\n<p",
       content);
printf(content, "%sThe answer is: %s\n<p",
       content, msg);
printf(content, "%sThanks for visiting!\n", content);

/* Generate the HTTP response */
printf("Content-length: %u\n", (unsigned) strlen(content));
printf("Content-type: text/html\n\n\n"), content);
Tiny Serving Dynamic Content

/* Return first part of HTTP response */
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Server: Tiny Web Server\r\n");
    Rio_writen(fd, buf, strlen(buf));

    if (fork() == 0) { /* child */
        /* Real server would set all CGI vars here */
        setenv("QUERY_STRING", cgiargs, 1);
        dup2(fd, STDOUT_FILENO); /* Redirect stdout to client */
        Execve(filename, emptylist, environ); /* Run CGI prog */
    }
    wait(NULL); /* Parent waits for and reaps child */

- Fork child to execute CGI program
- Change stdout to be connection to client
- Execute CGI program with execve
Web Proxies

- A proxy is an intermediary between a client and an origin server.
  - To the client, the proxy acts like a server.
  - To the server, the proxy acts like a client.

1. Client request
2. Proxy request
3. Server response
4. Proxy response
Why Web proxies?

- Can perform useful functions as requests and responses pass by
  - Examples: Caching, logging, anonymization, filtering, access control

![Diagram showing how Web proxies work]

- Fast inexpensive local network
- Slower more expensive global network
Conclusions

• Web serves generates output as response to client requests.
• The output generated can be static (displaying content from file) or dynamic (running a program to generate an output).
• There are many details here that are beyond the scope of this course.